Departament de Disseny i Programació de Sistemes Electrònics
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Prova final de TECPRO

19/06/2017 Grau en Enginyeria de Sistemes TIC 2H

COGNOMS:

NOM: GRUP de LAB:

Llegiu atentament els enunciats i responeu les preguntes que segueixen. Lliureu les respostes a cada exercici en fulls separats.

Exercici 1 [2 punts]. Teoria. **Apartat a)** Per cadascun dels enunciats següents, justifica si és [Cert/Fals]. (La no justificació invalida la resposta)

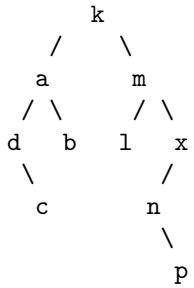
1. Si b i c són objectes de la classe A, i la classe A no ha realitzat una sobreescriptura/redefinició del mètode equals, llavors b==c evaluarà a True si i només si els valors de cada atribut de b i c són iguals.
2. Suposem les classes A i B, on s'ha definit la classe B com a public class B(A). Suposem que var és un atribut públic de la classe A. Suposem que la variable x és de tipus B. Llavors l'expressió x.var és correcta.
3. Cada mètode recursiu sempre necessita tenir almenys un paràmetre enter que es fa més petit cada vegada que es va cridant recursivament, i un cas base que el chequegi pels valors 1 i 0.
4. Donat un objecte del tipus de dades Stack, podem accedir als seus n elements, utilitzant el mètode top() n vegades.
5. Donada una llista ordenada, la cerca binària requereix cost logarítmic en cas pitjor.
6. Si una classe A té dos mètodes amb signatures $m(self, t = "")$ i $m(self, v = 0)$, diem que el mètode m està sobrecarregat.

Apartat b) Donada una llista de mida n, justifica quina serà el complexitat teòrica en cas pitjor, si ordenem la llista utilitzant les estratègies (La no justificació invalida la resposta):

1. selection sort
2. merge sort
3. bubble sort

Apartat c) Donat l'arbre binari que segueix, escriu-ne

1. el recorregut en pre-ordre,
2. el recorregut en post-ordre i
3. el recorregut en in-ordre.



Exercici 2 [3 punts]. Preguntes de resposta curta.

Apartat a) Escriu els resultats que es mostraran per pantalla.

```
def first(n):
    x = 0
    try:
        x = second(n)
    except StandardError:
        x = x+1
    return x

def second(n):
    y = 2
    try:
        y = third(n)
    except ValueError:
        y = y+5
    return y

def third(n):
    if n == 0:
        raise ValueError()
    elif n == 1:
        raise TypeError()
    return n+10

if __name__==’__main__’:
    print first(0)
    print first(1)
    print first(2)
```

Apartat b) Dissenyeu la funció booleana **recursiva** de nom *formatOK(input, labels)*, tal que, donat un input determina que aquest input està ben formatat respecte a una llista d'etiquetes proporcionada (labels), si i només si: 1) l'input és una llista, 2) l'input té com a mínim una longitud de 2, 3) el primer ítem de l'input es troba a la llista d'etiquetes i 4) cadascun dels ítems que queden a la llista és un string o bé una llista ben formatada.

```
def formatOK(input, labels):
    >>> formatOK([’VP’, [’V’, ’eat’]], [’VP’, ’V’])
    True
    >>> formatOK([’NP’, [’N’, ’a’, ’or’, ’b’], ’c’], [’NP’, ’V’, ’N’])
    True
    >>> formatOK([1, [2, ’oui’, [1, ’no’]], ’no’], [1, 2])
    True
    >>> formatOK([’VP’, [’V’, ’eat’]], [’VP’])
    False
    >>> formatOK([’VP’, [’V’]], [’VP’, ’V’])
    False
    >>> formatOK(’VP’, [’VP’, ’V’])
    False
    """
```

Apartat c) Donada la definició de la classe Persona que segueix, i que relaciona quina persona és compatible amb una altra,

```
class Persona(object):
    def __init__(self, n):
        self.name = n
        self.compatible = None
    def creaCompatible(self, p):
        self.compatible=p

    def __str__(self):
       #TO DO

def parellaCompatible(n1, n2):
  # TO DO
```

```
if __name__==’__main__’:
    s=Persona("Eva")
    print s #1
    t=Persona("Eric")
    s.creaCompatible(t)
    t.creaCompatible(s)
    print s #2
    print t #3
    l=parellaCompatible("Maria","John")
    for e in l:
        print e #4,5

#Resultats d’execució:
#1  Eva has no compatible person
#2  Eva has compatible Persona: Eric
#3  Eric has compatible Persona: Eva
#4  Maria has compatible Persona: John
#5  John has compatible Persona: Maria
```

c.1) Implementa el mètode *str* de la classe Persona, que permetrà el funcionament esperat detallat al joc de proves. **c.2)** Implementa la funció *parellaCompatible*, tal que, donats els noms de dues persones, crea

les 2 persones amb el nom corresponent, i fa les següents assignacions: la persona compatible de la primera és la segona, i la persona compatible de la segona és la primera. Finalment retorna la llista amb les 2 persones. **Apartat d)** Prenent com a base la classe desenvolupada en l'Apartat b), escriu un exemple de 1) mètode sobrecarregat, 2) mètode redefinit, 3) herència de mètodes i 4) delegació de mètodes.

Exercici 3 [3 punts]. Implementació de classes. iTICInsta està gestionant una aplicació per tal de gestionar els fans que segueixen un usuari, bloquejar-los si cal i gestionar els posts amb hashtags de les teves publicacions. A continuació et passen el prototipus de classes per iTICInsta per gestionar la informació i un exemple de funcionament. Cada Usuari té un nick, un conjunt de fans i un conjunt de posts. Cada post té un contingut i un conjunt de hashtags. Aquests hashtags poden incloure noms de nick (usuaris etiquetats).

```
class User(object):
    def __init__(self,nom,email):
        self.nick=nom
        self.email=email
        self.fans={}
        self.posts=[]
        self.__blocked=[]
    def __str__(self):
        #TO DO
    def addFan(self,p):
        self.fans[p.nick]=p
        #TO FINISH
    def checkFanInfo(self,fan):
        #TO DO
    def listFans(self):
        #TO DO
    def blockUser(self, user):
        #TO DO
    def listBlocked(self):
        #TO DO
```

Apartat a) Implementar els mètodes que permeten gestionar els usuaris fans d'un usuari i bloquejar-los si és el cas. A tal efecte, haureu de gestionar els mètodes

1. str: Mostra la informació completa d'un usuari i dels nicks dels seus fans
2. listFans: permet obtenir tota la informació dels fans d'un usuari
3. blockUser: ha de permetre bloquejar un usuari, tan si és fan com si no. Ha d'esborrar un fan de la collection de fans (si és fan). Caldrà afegir-ne el nick a la collection de fans bloquejats.
4. listBlocked: permet obtenir els nicks dels fans bloquejats
5. addFan: ha de permetre afegir un fan a la collection de fans de l'user. No es pot afegir un fan que s'ha bloquejat prèviament (per tant apareix a la collection de blocked de l'usuari en qüestió).
6. checkFanInfo: permet consultar tota la informació d'un fan de l'usuari que no s'hagi bloquejat, a partir del nick del fan. I el fan a consultar ha d'estar a la seva collection de fans.

```
if __name__=='__main__':
    g=User("pere","pere@itic.com")
    h=User("anna","anna@itic.com")
    l=User("maria","maria@itic.com")
    k=User("dimoni","dimoni@itic.com")
    g.addFan(h)
    print g #1
    g.addFan(l)
    g.listFans() #2
    g.blockUser(k) #3
    k.addFan(g) #4
    print k #5
    print g.listBlocked() #6
    g.checkFanInfo("maria") #7
```

```
#Resultats execució
#1  nick --> pere
    email --> pere@itic.com
    Fans_list_nicks --> anna
    No posts yet
#2  nick --> anna
    email --> anna@itic.com
    No fans yet No posts yet
    nick --> maria
    email --> maria@itic.com
    No fans yet No posts yet
#3  Not existing fan
#4  Not allowed
#5  nick --> dimoni
    email --> dimoni@itic.com
    No fans yet No posts yet
#6  Blocked users --> dimoni
#7  nick --> maria
    email --> maria@itic.com
    No fans yet
    No posts yet
```

Apartat b) Afegir el mètode $post(self)$ a la classe User, tal que permeti gestionar un post.

També us cal implementar el mètode *str* de la classe *Post*, per visualitzar-lo correctament.

Un post consisteix en un string, el qual conté a la part final els hashtags amb la sintaxi #nomHashtag i els usuaris etiquetats amb la sintaxi @*usuari*. A continuació segueixen els jocs de prova.

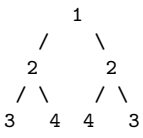
Es pot etiquetar qualsevol usuari (no cal que siguin fans), excepte aquells que hàgim bloquejat.

```
import time
class Post(object):
    def __init__(self,text,date):
        self.content=text
        self.date=date
        self.hashtags=[]
        self.friends=[]

    def __str__(self):
      #TO DO

    def addHashtag(self,h):
        self.hashtags+=[h]

    def addFriends(self,f):
        self.friends+=[f]

class User(object):
    def post(self,p):
       dataActual = time.strftime("%c")
       #TO FINISH

    def printPostsDate(self):
        for p in self:
            print p

if __name__=='__main__':
    g=User("pere","pere@itic.com")
    l=User("maria","maria@itic.com")
    g.addFan(l)
    g.blockUser(l)
    g.post("Next starting project #newproject #newversion #workingcopy @eva @maria @jordina")
    g.post("Holidays are comming #finishing_project")
    g.post("End of exams @jordi @eva @maria")
    g.post("no way")
    g.printPostsDate()

#Resultats d'execució
Content -->Next starting project
Date--> Mon Jun 12 10:29:47 2017
Hashtags--> #newproject  #newversion  #workingcopy
Labeled users--> @eva @jordina

Content -->Holidays are comming
Date--> Mon Jun 12 10:29:47 2017
Hashtags--> #finishing_project

Content -->End of exams
Date--> Mon Jun 12 10:29:47 2017
Labeled users--> @jordi @eva

Content -->no way
Date--> Mon Jun 12 10:29:47 2017
```

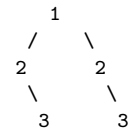
Exercici 4 [2 punts]. Binary Tree. Forest Gump t'ha llogat per chequejar si els seus arbres binaris són mirall/simètrics. 2 arbres/subarbres són mirall/simètric,

- si tenem el mateix valor per l'arrel, i
- el subarbre esquerre de l'arbre esquerre i el subarbre dret de l'arbre dret són mirall, i
- el subarbre dret de l'arbre esquerre i el subarbre esquerre de l'arbre dret són mirall.

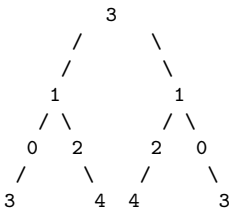
El següent arbre BT és simètric



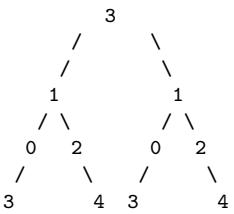
El següent arbre BT no és simètric



El següent arbre BT és simètric



El següent arbre BT no és simètric



La classe de partida i el joc de proves proporcionats són els que segueixen. Chequegeu-los amb cura per entendre el seu funcionament.

```
class BT(object):
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

def isMirror(tree):
    #TO DO
```

```
if __name__==’__main__’:
    root = BT(1)
    root.left = BT(2)
    root.right = BT(2)
    root.left.left = BT(3)
    root.left.right = BT(4)
    root.right.left = BT(4)
    root.right.right = BT(3)
    print "1" if isMirror(root) == True else "0" #escriurà 1
    root = BT(1)
    root.left = BT(2)
    root.right = BT(2)
    root.left.right = BT(3)
    root.right.right = BT(3)
    print "1" if isMirror(root) == True else "0" #escriurà 0
```

[Apartat a] A tal efecte cal que dissenyeu i implementeu únicament la funció **recursiva** *isMirror*, tal que ha de comprovar si el BT és simètric/mirall o no ho és.

[Apartat b] Trieu i **justifiqueu** quina és la complexitat d'aquesta funció respecte al número de nodes de l'arbre (n).

1. Linial $O(n)$
2. Logarítmica $O(logn)$ o bé $O(nlogn)$
3. Quadràtica $O(n^2)$
4. Exponencial $O(2^n)$